Practical Computer Security through Cryptography

A. David McNab[†]
NAS Technical Report NAS-98-015

NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000
mcnab@nas.nasa.gov

Abstract

The core protocols upon which the Internet was built are insecure. Weak authentication and the lack of low level encryption services introduce vulnerabilities that propagate upwards in the network stack. Using statistics based on CERT/CC Internet security incident reports, the relative likelihood of attacks via these vulnerabilities is analyzed. The primary conclusion is that the standard UNIX BSD-based authentication system is by far the most commonly exploited weakness. Encryption of sensitive password data and the adoption of cryptographically-based authentication protocols can greatly reduce these vulnerabilities. Basic cryptographic terminology and techniques are presented, with attention focused on the ways in which technology such as encryption and digital signatures can be used to protect against the most commonly exploited vulnerabilities. A survey of contemporary security software demonstrates that tools based on cryptographic techniques, such as *Kerberos*, *ssh*, and *PGP*, are readily available and effectively close many of the most serious security holes. Nine practical recommendations for improving security are described.

[†]David McNab is an employee of MRJ Technology Solutions, Inc.

1/ Introduction

Internet protocols were designed to interconnect geographically distant computers, ultimately to help people share information. Their designers, working in an environment where this was awkward at best, were primarily concerned with removing obstacles to information sharing. The network community was tiny by today's standard and was mostly populated by professionals and academics with a vested interest in cooperation. Given this design context, it is not surprising that security was a minor concern.

Compared with the original environment, today's Internet is egalitarian. The ivory tower has been thrown down, and along with the positive and sometimes amazing results, serious problems have appeared. One of these is the exposure of fundamentally insecure network protocols to a subset of the on-line community only too happy to exploit security vulnerabilities. The natural opposition between sharing information with one's allies and protecting it from one's enemies—or at least the uninitiated—is played out in a continual battle of technological leapfrog. New vulnerabilities are uncovered only to be closed by new tools, some of which inevitably introduce new vulnerabilities.

The security holes most easy to exploit are those closed first—at least at computing sites vigilant to security threats. "Crackers" are then forced to seek new modes of attack. There are two general fronts. The first is at the application level: almost every networked application introduces security concerns, if not outright vulnerabilities. As more of these applications are developed and incorporated into the typical computing environment, the number of vulnerabilities increases. In addition, the more complex the software the more likely it is to be misconfigured, leaving open an avenue of attack. This can be considered the "widening" front of attack. Here the game of leapfrog is played out at its most obvious, as new applications are released, attacked, patched, the holes opened by the patches attacked, and so forth.

The other direction of attack is the "deepening" front, where the attacker takes advantage of his or her specialized knowledge and access to more powerful technology to attack "lower" parts of the network infrastructure. This type of attack has been made easier by the introduction of network interfaces to Windows and Macintosh computers, and by the increased availability of high quality free UNIX implementations—complete with easily buildable source code. This report focuses on the "deepening" front of attack and the tools that are available to defend it.

The lowest level of Internet infrastructure, the IP protocol, is fundamentally insecure. There is no provision for cryptographically-based authentication nor for encryption of protocol fields. (Data payload can be of arbitrary type and format, hence can be encrypted.) This has motivated the development of a secure version of IP, called IPsec, now being tested and standardized[12]. IPsec provides encryption and authentication services for the most fundamental Internet software protocol[13, 14]. Unfortunately it will be some time until IPsec is well deployed, and until then IP will continue to be vulnerable.

Despite IP's intrinsic vulnerability, one can use it as a base upon which to construct secure higher level protocols. For example, applications can implement their own authentication and encryption protocols encapsulated in IP packets. Computers running the more secure protocols are still open to attacks at the IP level, but users can choose to take advantage of the higher level protocols. This allows individuals to be more confident of the integrity and privacy of their data, despite the continuing risk of some types of system-wide attacks. In addition, by taking away the "easy" vulnerabilities, an administrator can make his or her system less attractive to prospective attackers, who may simply choose a less secure system as a target.

2/ Overview

This paper focuses on the vulnerabilities resulting from IP's fundamental security weaknesses and introduces technology that can be used to build more secure higher level protocols.

The first section discusses the potential problems resulting from lack of encryption and poor authentication. These are extant weaknesses in the Internet protocols and application suites that can be exploited to compromise security.

Next there are statistics drawn from Dr. John Howard's PhD dissertation, "An Analysis of Security Incidents on the Internet 1989-1995"[1]. Howard's work is a comprehensive analysis of security incident reports to the Computer Emergency Response Team Coordination Center (CERT/CC) at Carnegie Mellon University, which has collected security incident reports from its inception in 1988. Howard's statistics and analysis provides a useful gauge of the actual threat posed by the theoretical vulnerabilities.

Following the overview of Howard's work are descriptions of technology that can be used to close the most severe vulnerabilities. The discussion is largely theoretical, leading to the next section where attention shifts to contemporary implementations of the theory. Finally there is a series of practical recommendations for improving security.

3/ POTENTIAL VULNERABILITIES

The IP-based network protocols and the applications built on them are ubiquitous on the Internet today. Examples of IP-based protocols are TCP, UDP, FTP, and NFS. IP-based applications include interactive session tools like *telnet*, *rlogin*, and *rsh*; data transfer tools like *rcp* and *ftp*; windowing systems such as *X11*; and electronic mail infrastructure.

All of these tools depend on IP for their lowest level of software networking and exhibit security weaknesses inherited from IP, for example susceptibility to IP spoofing. Some introduce further vulnerabilities due to their own flaws, for example by transmitting cleartext passwords. This section discusses the basic vulnerabilities that threaten sites relying on standard IP-based networking tools.

Most people, if they even stop to consider it, imagine that computer intrusions originate outside the target site. The mass media driven stereotype suggests that the "hacker" is a young man with poor personal hygiene habits and a socio-pathic desire to annoy corporate and government computer administrators. In truth, as described in § 4.1, there is little documentation illuminating the nature of attackers. The data that does exist suggests roughly an equal risk of attack by an insider—for example a disgruntled former employee—and attack from outside. (The exact statistics are presented in § 4.) Of course, once an external attacker gains access to the internal network, he or she becomes an internal attacker. (For these reasons, firewall-based defenses, which encourage the illusion that the outside world is evil and hostile but the internal network is innocuous, can be profoundly dangerous.) In the discussion of potential vulnerabilities, cases where internal attacks present a particularly severe or unexpected threat are annotated.

3.1/ PASSWORD SNIFFING

Any time a password is transmitted over the Internet, for instance in response to a login challenge from *telnetd*, it is vulnerable to interception. Furthermore, there is virtually no way to know that the data has been intercepted. Any computer attached to any *unswitched* network between the user and the target host is capable of reading the transmission stream and extracting the password. The majority of local area networks deployed today are unswitched. For example, standard Ethernet is essentially a "party line". Any data transmitted to any host on the local network are available to all other local hosts; the operating system simply chooses to accept only those packets with the "correct" IP address in the protocol header. Nonetheless, commonly available network diagnostic tools (such as *tcpdump*) include options that allow full access to the network stream, including packets destined for other machines. There are options to target specific hosts, or specific protocols, or combinations of the two. On the majority of UNIX machines this interception requires the root password, but this is an operating system policy rather than a security feature of the network protocols.

There are *switched* networks available; in fact there is a switched version of Ethernet. A switched network essentially builds a private connection between two (or more) hosts, rather than broadcasting all messages on the party line and letting the hosts pick out the ones intended for them. This offers a number of advantages, including improved security.

This paper focuses on the problems associated with unswitched networks, primarily because they are currently the most common. For a few sites, converting to a switched LAN is a valid option. In many other cases, however, the cost or inconvenience is prohibitive. Furthermore few system and security administrators have the luxury of modifying the physical network infrastructure—and if they do, the process is likely to be time consuming. Finally, even if a site's internal networks are fully switched, external traffic is likely to flow over an unswitched network. For that reason alone, all administrators should understand the related security implications.

A password interception scenario might begin with the compromise of an Internet Service Provider (ISP), either by an external cracker or by unethical monitoring performed by a rogue ISP employee. Any customer of that ISP who connects to remote computers using an unencrypted password exchange—e.g. using *telnet* or *rlogin* without a *.rhost* file—compromises his or her password by typing it. The cracker monitors network traffic, looking for newly created sessions on the ports used for login and watching for the characteristic "`Password:`" prompt and its response. Unless the attackers then uses the password in a foolish, easy to detect way, the user has given away access to his or her account without even knowing it.

A more serious scenario is that after connecting to the remote machines the victim types the root password, perhaps in response to an *su* prompt to do some administrative work. This is a dire situation—undetected compromise of the root password and the password of a user with legitimate root privilege.

Typing a root password over an unencrypted remote connection is worse than writing down the password and then losing it—worse because at least in the latter case the user knows that he or she may have compromised the password, and because somebody who finds the dropped password note has no way of knowing for which machine it is useful. Interception of a cleartext root password is undetectable and implicitly identifies the compromised machine. In a security environment that depends entirely on passwords for authentication, this level of risk is clearly unacceptable. Luckily, it is also one of the easiest vulnerabilities to correct, as discussed later.

If the attacker has physical access to the target's internal network from a computer to which he or she has root access—e.g. a laptop—all unencrypted traffic on that subnet is compromised. Furthermore, even wary, security-conscious staff will tend to trust the internal network, and many will not think twice about typing a root or personal password over an internal network connection—say from a desktop machine to a server.

In the worst case, an administrator with legitimate access to one or more root passwords is the attacker. If the password is associated with a shared host, for example a supercomputer or a general purpose email host, the attacker can gather root and personal passwords by directly snooping on local login sessions (e.g. by snooping the kernel `clist` data structures). This type of attack can also occur when an external attacker learns the root password. In either case, the attacker can easily build a substantial list of legitimate passwords.

3.2/ TRUSTED HOST COMPROMISE

Another straightforward vulnerability involves the compromise of a generally trusted host. This is usually an internal machine, but could also be situated off-site. For example, a supercomputer center may implicitly trust a machine situated at a university or a government laboratory. Anybody with root access to that machine can locally "sniff" any password typed there, for example in response to the supercomputer center's *telnet* or *su* password queries.

This is very similar to the network sniffing attack, except that it is more difficult to prevent. An encrypted login session, for instance, can protect against interception along the session's route, but sniffing on the client side examines the typed data before they are encrypted. On the other hand this vulnerability is inherently a smaller risk, because it requires compromise of an endpoint machine instead of one of many on a network route. From the attacker's point of view, the intermediate machines are much more desirable targets, because they usually carry more traffic and hence many more passwords.

Another mitigating factor is that endpoint machines are generally better monitored and more difficult to compromise. It is relatively hard to detect suspicious activities on busy ISP host, especially when compared with a desktop machine "owned" by an individual. Nonetheless, local host sniffing attacks are a substantial threat and should be addressed.

3.3/ Spoofing

Spoofing, in general, is an attack based on assuming a host identity to which one is not entitled. TCP/IP tools conventionally determine the identity of a remote host by looking at its network address. In other words, the server assumes that the remote client is entitled to make use of its network address and that the packets seeming to come from that address actually do. This is a painfully naive assumption, perhaps defensible in the mid-eighties when TCP/IP-capable computers were generally expensive machines under professional control, but certainly an unacceptable risk in today's environment of inexpensive UNIX-capable portable computers. All an attacker has to do is look for a trusted remote host that has crashed or been taken down for maintenance, then run a few configuration commands to assume its identity. Since the attacker has full privileges on his or her own machine, it is then trivial to assume the identity of a legitimate user of the spoofed machine and to gain access to a remote host that trusts it.

There are more sophisticated spoofing attacks. For example, Domain Name Service (DNS) spoofing. This is an indirect attack on the mechanism used to authenticate BSD-style *rcmds*[1]. The most common use of the *rcmd* authentication mechanism involves comparing the hostname of the originating connection against an access control list: if the hostname is on the list, access is permitted. However the incoming connection does not actually carry a hostname; it carries an IP address. Host names are mapped to addresses using a name resolution scheme, generally DNS. If an attacker can illegitimately alter the DNS *(name, address)* mappings so that the attacker's machine address resolves to a trusted host's name, he or she gains access to the target machine.

Alternatively, an attacker could corrupt the DNS database of the originating host, so that a commonly used remote hostname is mapped to an IP address belonging to an attacker-controlled machine. A local user trying to connect to that host naively types his or her password; the attacker reads it, stores it, and then uses it to transparently set up a connection with the legitimate target machine. The end result is a compromised password and only a slim chance of the user detecting a well implemented attack.

If an attacker has physical access to a workstation connected to the internal network, he or she can easily unplug that machine's network cable and plug in a laptop. It is then possible to masquerade as the workstation knowing that there is no possibility of address collision. If the attacker is a former or current employee and is familiar with local staff and policies, he or she will be able to target an attack to gain the maximum advantage.

[1]The *Rcmds* are Berkeley Software Distribution network clients such as *rlogin*, *rsh*, and *rcp*, all of which employ an underlying networking library function called *ruserok()* for authentication.

3.4/ Physical Security

As previously mentioned, having physical access to computers on the target network makes some attacks much easier (e.g. spoofing). In an egregious case, for example if the attacker happens across an unprotected machine where a legitimate user is logged in, a number of nasty opportunities exist. Easy ones include copying the password file for later cracking, inserting Trojan horse programs to capture passwords, and of course the compromise of data private to the legitimate owner of the account. The effects of the latter could range from embarrassment to corporate espionage to exposure of additional security vulnerabilities.

Although stumbling across an active, unprotected login session would be serendipitous, there are equally useful and less haphazard ways to gain access. For example, some older systems provide the option of a boot into privileged single user mode without requiring a password. The attacker can reboot the machine into single user mode, create a set-UID root shell, for instance, and then complete the boot into multiuser mode. This gives root access to the machine. Other attacks include the use of "magic" key combinations to force reboots or to terminate the X server, which can sidestep a screen lock. If the X session was started from a terminal login, rather than by *Xdm*, this can give access to a user's account.

Even if machines are well defended against this type of attack, the network itself is vulnerable to physical compromise. It is a simple matter to slip into an unoccupied cubicle—perhaps at lunch time—and plug a portable computer's Ethernet cable into a wall jack. At this point there are any number of sniffing and spoofing vulnerabilities to exploit, most of which are similar to their external counterparts, but exacerbated because the attack is originating from what appears to be a trusted internal host.

3.5/ Shared Logins

Shared logins are a significant security vulnerability, for several reasons. The first is that they reduce accountability. If somebody tampers with a personal account, the tampering is usually discovered by the legitimate account owner, because he or she recognizes some irregularity—for example, a "last login" message indicates login from a host unknown to the user, or a new file or directory suddenly appears. A shared login is not "owned" by any one person, and so these events are likely to pass unnoticed. If a shared account is compromised, the activity of the attacker will be difficult to reconstruct, because other legitimate users will disturb the evidence.

If the account is shared among many people or is frequently used, it is particularly dangerous. For example some sites have an "operations" account that allows their administrative staff access to all computer systems. By cracking this account, an attacker gains access to every machine at the site. Nobody is likely to recognize illegitimate use, as mentioned above, and nobody will think twice if they see this user *su* to root. Furthermore, the more an account is used, the more chance there is that it will be cracked in the first place—i.e. the more opportunities there are to stumble across a forgotten login session, or the more times the cleartext account password traverses the network. Thus the chances of an attacker gaining access to a busy account are significantly higher than to a personal account. Combining this with the difficulty of detecting such a compromise, and the degree of access it affords, it should be clear that this type of shared account should be avoided.

3.6/ Email Forgery

Forging email is trivial, and if the attacker composes the message carefully the forgery may be very difficult to detect. This is especially true if the attacker knows something about local procedures and staff assignments, which he or she can learn from a variety of sources—for example by careful reading of the site's support web pages, or by intercepting clear text messages on the local nets.

Usually, forged email is only indirectly useful to an attacker. For example, one could send a message asking an administrator to do something that requires root access, and then monitor that administrator's network traffic to try to intercept a root password. But a more bold approach might work too. For example, if the "owner" of a desktop workstation asks that another user be given an account there, the account may be created almost immediately and without any further authorization. If an attacker has cracked an otherwise inactive account but wants access to a different subnet, he or she could forge a message asking for an account on an appropriate machine. If done carefully, this could pass without notice and would probably not be detected quickly.

Email forgery is particularly difficult to detect and trace if it originates from a local host or a machine spoofing a local host. Disgruntled employees have the added advantage of being intimately familiar with local staff roles and procedures, which helps to reduce the chance of detection because of a content irregularity.

3.7/ INDIRECT VULNERABILITIES

A more subtle vulnerability results from the routine use of cleartext network transmissions. Although the information contained in the transmissions may not seem sensitive, it can sometimes be used to an attacker's advantage. The apocalyptic example is that journalists were supposedly able to correlate the beginning of major U.S. military operations with dramatically increased pizza deliveries to the Pentagon. The risks resulting from this type of traffic analysis attack often seem far-fetched and not particularly threatening, but in fact the monitored information can suggest an easy avenue of attack. For example, an attacker can see in a mundane email message that a certain host will be down for some period of time, making that host a prime target for spoofing. Or perhaps the attacker observes that certain network addresses are reserved for temporary or test use, and thus are "expected" to behave oddly.

4/ STATISTICAL THREAT ASSESSMENT

A system administrator reviewing the list of potential security vulnerabilities could well feel daunted. To address them all requires substantial effort, and perhaps more importantly may end up interfering with users' legitimate activities. Most administrators will be willing to trade-off a small amount of risk for a substantial reduction in work and cumbersome restrictions. To do so intelligently, an administrator should understand not only the consequences of each type of attack, but also the likelihood that it will be undertaken. Historically, the former information is easy to come by, usually included with security vulnerability alerts, whereas the latter is very difficult to find.

Dr. John Howard's Carnegie Mellon University PhD dissertation supplies a statistical gauge of the prevalence of various types of attack. Howard analyzed security incident reports gathered at CERT/CC, the Computer Emergency Response Team Coordination Center. CERT/CC was established by the Defense Advanced Research Projects Agency (DARPA) in November 1988, within weeks of the notorious "Internet Worm" incident, and serves as a clearinghouse for computer security information. Howard developed a taxonomy of computer security attacks and used it to organize the CERT records from 1989 to the end of 1995.

Howard describes his taxonomy as "operational" and "process based": *attackers* use *tools* to gain *access* of a certain type and via certain a *vulnerability*. This access has *results* that are presumed to be useful in achieving one or more *objectives*. Howard examined CERT's logs and extracted information appropriate to each of these categories. He then statistically analyzed the results.

A CERT report describes a security *incident*. One incident may involve several *attacks*—in some cases a large number. An incident report also lists any exploited vulnerabilities, any tools that may have been used, and the results of the attacks. However there is no direct association between these data points and specific attacks, just a general association with an incident. Thus Howard is unable to statistically analyze the nature of attacks; he is forced by the source data to work with incidents, which may each include several vulnerabilities, tools, or results. One consequence is that a single incident may be counted in several categories, so that in some cases the number of incidents associated with sub-categories may total more than the overall number of incidents.

One caveat to be considered when reviewing Howard's data is that the source records are incomplete relative to his taxonomy, and CERT/CC's objectives in gathering them are different from Howard's analysis objectives. For example, CERT/CC places much more emphasis on the vulnerabilities that are being exploited than the objectives or nature of the attackers. This leads to a paucity of information in some of Howard's taxonomical groups.

In many cases it seems reasonable to extrapolate results to the Internet population as a whole, but this is not necessarily valid. Readers interested in a detailed discussion are referred to the dissertation itself[1].

Despite these minor issues, Howard's work is valuable. The caveats described above are due to the limitations in the source material, rather than his analysis. In the past, most "threat models" have been largely speculative, based on a small amount of local data at best. Howard does a fine job of analyzing a significant population of security incidents and providing a quantitative basis for a threat model. Perhaps the most important information is the *relative* frequencies of incident types, vulnerability exploitations, and tools used.

4.1/ ATTACKERS

Unfortunately there are few data in this area. 35 of 4,299 incidents reported to CERT/CC included the identity of the attacker. It appears that in many more cases the identity of the attackers was determined but not reported. Howard speculates that this is due primarily to CERT/CC's mode of operation, in which attention is focused on events as they occur. By the time the activity has stopped and the target site's administrators begin to track down the intruder, there is less involvement with CERT/CC. Another possibility is that many "attacks" are mischievous rather than malevolent, and that reporters preferred not to subject the culprits to possible punishment.

Of the 35 incidents that do include the attacker's identity, 18 involved attacks by external "crackers" and 17 were initiated by former employees. However it is not clear whether the attacks initiated by former employees necessarily occurred from within the target site's computing environment, nor that external crackers necessarily attacked from the outside. It would be unwise to base strong conclusions on such a small and incomplete sample, but clearly it is prudent to assume that there is a legitimate internal threat as well as the more widely publicized external one.

4.2/ TOOLS USED

Howard's *tools* category attempts to determine methods of attack. Only 18.1% of incidents reported to CERT referred to the use of one or more tools to gain access. The most popular are listed in Table 1. The percentages are fractions of the number of incidents that mentioned tools, rather than of all incidents (i.e. percentages of the 18.1%). Also note that as mentioned in the introduction to this section, the tool categories are not mutually exclusive: for example, reports that mention the use of Trojan horses often mention an attack on more than one program. This means that the sum of the number of incidents mentioning specific tools is greater than the total number of incidents for which the use of at least one tool was reported.

An additional complication particular to the *tools* category is that more sophisticated tools necessarily involve the use of less sophisticated ones. For example, a toolkit designed to break into root accounts may include scripts and programs as well as user commands that initiate them. Rather than describe this incident as using all four types of tool, Howard counted it as the most sophisticated category—in this case as the use of a toolkit, not a program, script, or user command.

% Tool Incidents	Tool Used
57.8%	Trojan horse
56.0%	login
15.6%	telnet
11.8%	ps
16.6%	other (42 programs)
31.2%	Sniffers
23.8%	Toolkits[a]
14.3%	scanners (e.g. ISS, SATAN)
9.9%	targeting root (e.g. rootkit)
7.9%	Password cracking tools

[a]Toolkits are pre-packaged collections of scripts and programs that are useful for breaking into computer systems, often including instructions and suggestions for erasing evidence of the attack.

Table 1: Most Common Cracking Tools

For our purposes, the most significant entry in Table 1 is for *sniffers*: tools that monitor network transmissions and are capable of intercepting cleartext passwords or other sensitive data. Almost a third of the incidents reporting the use of a tool mentioned the use of a sniffer, and according to Howard the use of sniffers is increasing (as discussed later). The other obvious observation is that Trojan horse attacks are very popular. This is of less concern, primarily because Trojan horses can be effectively found and neutralized using readily available checksum monitoring programs. The "toolkits" category included tools designed to acquire root privileges, primarily *rootkit*, and "scanners", such as *ISS* and *SATAN*.

4.3/ Type of Access

Howard defines a *type of access* category in his taxonomy. This describes the level of access gained and gives an indirect idea of the objectives of the attacker. Again, it is not immediately clear how to categorize an incident, since it may involve several attacks, each resulting in a different type of access. Howard's resolution is to categorize an incident based on the most severe type of access, according to an intuitively reasonable hierarchy of severity. Table 2 summarizes the data.

% All Incidents	Type of Access
89.3%	Unauthorized access
37.5%	failed attempts
27.7%	to root accounts
26.9%	to other accounts
10.7%	Unauthorized use
5.1%	disclosure of information[a]
3.1%	corruption of information[b]
2.4%	denial of service

[a]approximately 80% of disclosure incidents involved the use of anonymous ftp for the deposit or transfer of pirated software

[b]all of these incidents were classified as IP spoofing; approximately 95% involved email source spoofing and the remainder were other packet source spoofing incidents

Table 2: Types of Access

The first observation is that roughly 90% of incidents reported to CERT/CC involved attacks aimed at gaining access to the target machine. The count of failed attempts is probably low, since they are much less likely to be reported. A more interesting point is that the root account is as likely to be the target of a successful attack as a user account. Again the statistics may be misleading, since attacks on the root account are more serious and probably more likely to be reported. Furthermore if a single incident involves the compromise of ten user accounts and the root account, it is counted as a root account break-in because this is the most serious type of access.

The remaining 10% of incidents involved the misuse of a computer system to which the miscreant had legitimate access. Taking into account the table footnotes, the three dominant misuses are staging for software piracy, email source spoofing, and denial of service attacks. The latter are generally very difficult to protect against, and although they are fairly easy to neutralize once they happen, they are inconvenient enough that most administrators will be glad to note their relative infrequency. For our purposes the most interesting information is the frequency of email spoofing: essentially 3% of all incidents reported to CERT/CC.

4.4/ EXPLOITED VULNERABILITIES

Roughly half of the CERT reports (45.3%) specifically mention one or more vulnerabilities that were exploited by attackers. The most common vulnerabilities are listed in Table 3. The percentages represent the proportion of incidents mentioning vulnerabilities, not of all incidents. As with the "tools" report, many incidents mention multiple vulnerabilities.

% Vulnerabilities		Exploited Vulnerability
48.1%		Password problems[a]
22.9%		Sendmail
12.8%		Trusted hosts
	10.8%	.rhosts
	2.7%	hosts.equiv
12.8%		"Improper" configuration[b]
12.2%		TFTP [c]
10.8%		Mail spoofing
8.7%		FTP
7.1%		NFS
5.3%		Sun NIS
3.5%		Sun YP

[a]For incidents describing password vulnerabilities, 63.1% mentioned password file compromise (usually copying), 47.8% mention that a password was cracked, and 16.6% mention that a weak password was easily guessed.

[b]Refers to cases where network software was clearly misconfigured; see the text for more explanation.

[c]TFTP vulnerabilities became well known during the reporting period, and statistics indicate that the rate of incidence dropped over time, suggesting that they were being corrected. However a few incidents continued to occur.

Table 3: Vulnerabilities

Almost 50% of vulnerabilities that provided illegitimate access involved passwords. According to the CERT data, roughly the same number of incidents were those in which a password was thought to have been cracked. This seems speculative, since there is little to demonstrate how the incident reporters could make that determination. To do so conclusively would be quite difficult: an administrator would have to find a "smoking gun", i.e. a password cracking program running on the target machine. However, password cracking tools were mentioned in only 52 reports, compared with 448 in which password cracking was mentioned.

Perhaps a safer conclusion would be that of incidents mentioning the exploited vulnerability, 23% involved the use of a password thought to be secure. A separate category, which likely has some degree of overlap, is weak passwords: 8% of incidents mentioning a vulnerability also mentioned a weak password thought to be easily guessable. In either of these cases snooped IP data could lead to the same result; it is unclear why one should assume that brute-force cryptanalysis occurred.

Also among the top four vulnerabilities is the exploitation of trusted hosts systems, i.e. the *ruserok()* based authentication system. Combine this with the exploitation of passwords as previously discussed, whatever the details leading to the discovery of the password, and it is clear that the vulnerabilities of the base UNIX authentication system are being actively exploited.

The "improper configuration" category addresses the proposition that the majority of break-ins result from operator error, i.e. that the target system is misconfigured and allows open access through some network service. The CERT/CC records suggest otherwise, since fewer than 13% of the compromises in which the cause was identified indicated misconfiguration. Nonetheless it is a significant problem, particularly since network software is increasingly complicated hence more prone to misconfiguration.

The incidents mentioning *sendmail*, *TFTP*, *FTP*, *NFS*, and Sun distributed password systems indicate that application-level vulnerabilities continue to be a significant problem.

Mail spoofing also appears as a common vulnerability. This is somewhat puzzling, as mail spoofing is unlikely to lead to unauthorized access. Furthermore, there is a numerical discrepancy. According to Table 3, 210 incident reports mentioned email spoofing. However Table 2 and its footnote state that 95% of 3.1% of corruption of information incidents, or only about 127 reports, involved email source spoofing. Most likely, the remaining email spoofing cases were mentioned in an incident report that also clearly described an access event and hence was sorted into the more serious *unauthorized access* group.

4.5/ Trends in Severe Incidents

Howard developed criteria by which he classified a few of the 4,299 reported security incidents as *severe*. Choosing criteria was not a trivial procedure, since there was no clear single factor that could be used for discrimination. Howard's statistical reasoning is not reproduced here, but the final result was the selection of 22 severe incidents, each of which lasted at least 79 days, involved at least 62 sites, resulted in at least 87 messages to CERT/CC, and involved a root break-in.

Of the 22 severe incidents, 21 appeared to be three phase attacks. First, the attacker established an account on the target machine, through password cracking, sniffing of cleartext passwords, or other means. Next, the attacker exploited local vulnerabilities to gain root privileges. Finally the compromised machine was used as a staging point for attacks on other systems. The one exception to the three-phase pattern was an incident involving IP spoofing.

Early in 1994 there was a brief period during which no severe incidents were occurring. This time was an epoch of sorts: the incidents occurring afterwards had a clearly different character from those before. Prior to the epoch there were ten incidents, six classified as using command line tools, simple scripts, and password cracking, two involving sniffers, one involving TFTP attacks, and one case of FTP abuse for software piracy. After early 1994, command line based attacks no longer occurred. Of the twelve severe post-epoch incidents, eleven involved sniffers, and six of those also involved the use of toolkits.

The remaining incident, in 1995, involved IP spoofing. The attackers essentially built IP packets to order, inserting whatever originating address they desired. Since the *ruserok()*-based authentication used in most UNIX networking utilities relies on source address to determine the trustworthiness of the user identification, this gave attackers access to the target machine. In some cases root access was gained directly.

4.6/ Summary and Conclusions

The vast majority of reported security incidents involved attempts to gain unauthorized access to target computers. Almost one third of these resulted in the compromise of the root account. More than half of access attacks exploited vulnerabilities in passwords and the trusted host system—in other words they attacked the basic authentication system used in BSD-based UNIX networking. The remainder attempted to exploit network application weaknesses, particularly in *sendmail* but also in commonly used tools like *FTP* and *NFS*.

The trend in severe incidents is clearly towards the use of sniffers, which can pick passwords out of cleartext network transmissions, and other low level IP attacks, such as spoofing and connection hijacking.

The conclusion is that the current widespread reliance on passwords and IP packet source addresses for user authentication is increasingly dangerous, particularly when passwords are transmitted in cleartext by standard network applications. System administrators concerned about security should be looking to transmission stream encryption and cryptographically strong authentication protocols to reduce their vulnerability. The next section is a discussion of technology that can be used for these purposes.

Although the trend in severe incidents is towards "deeper" attacks, application-level vulnerabilities are still being exploited, and attackers are increasingly doing so using automated scanners and toolkits that can probe a large number of systems and vulnerabilities very quickly. Luckily these weapons can also be used by administrators to probe their own systems, so that the holes can be closed before an attacker finds them. The best defense against this class of attacks is a thorough and continuous program of self-inspection.

5/ Defensive Technology

This section reviews technology that can be incorporated into security tools to help close some of the vulnerabilities just described. Later, tools that actually implement some of these ideas are introduced.

The focus is on cryptography, because it is the most powerful and versatile protection against sniffing, IP-spoofing, and other "low-level" attacks.

Application level attacks are best prevented with automated self-scanners and a reasonable level of vigilance. There are a number of tools available to perform this scanning—in many cases the same tools used by attackers—and to automate procedures that allow an administrator to maintain a high level of vigilance without investing too much time. These types of tools are not discussed in this paper; the focus is on the "deepening" front of attack.

5.1/ Basic Cryptography

Ordinary, unencrypted data is called *cleartext*; encrypted data is called *ciphertext*. The mathematical algorithm that converts cleartext to ciphertext (and vice versa) is called a *cipher* or *cryptographic algorithm*. The implementation of the cipher is a *cryptosystem*. A person who studies ways of effectively encrypting data is a *cryptographer*, whereas a person who specializes in extracting cleartext from the ciphertext—in other words in breaking a code—is a *cryptanalyst*.

Ciphers employ a *key*: a value used to specify some internal details of the cipher's operation. To decrypt a message, one needs to know both the cipher and key that were used to generate it. The possible range of values of the key is called the *keyspace*, and the cipher is most secure when the keyspace is very large. If the keyspace is too small, a cryptanalyst can exhaustively search it to determine which key will decrypt the ciphertext.

The ciphers of most concern fall into two categories. *Symmetric* ciphers are those for which only a single key is necessary. Ciphertext generated by a particular key can be decrypted using the same key[2]. In order to use a symmetric cryptosystem for communication, both the sender and the recipient of a message must know the key. In addition, the key must be kept secret if the communication is to remain secure. For this reason, symmetric ciphers are often called *secret key* algorithms.

Most symmetric ciphers are also *block* ciphers, meaning that they encrypt data in chunks known as blocks. There are also *stream* ciphers, which encrypt data continuously as it "flows through" the algorithm.

An *asymmetric* cipher, on the other hand, is one that employs two "linked" keys, called a *key pair*. Knowledge of either key does not provide any useful clues about the value of the other key, and data encrypted with one key can only be decrypted with the other. Generally one key is kept secret and is known as the *private key*, and the other is freely available and called the *public key*. The great advantage of this type of cryptosystem is that the public key can be advertised freely, so that anybody wishing to communicate securely with the private key holder need not be trusted. In part because of this, asymmetric cryptosystems are also known as "public key" systems. The primary disadvantage of this approach is its speed: asymmetric algorithms are typically three orders of magnitude slower than symmetric ones.

Hybrid cryptosystems use both types of cipher in concert, in an attempt to reap the advantages of both without incurring the disadvantages. Typically a public key system is used to securely exchange a symmetric key at the beginning of communications. The symmetric key, or *session key*, is then used to encrypt the subsequent data exchange. Thus the key management advantages of

[2]Technically, the definition of a symmetric cipher is broader. The encryption and decryption keys do not have to be the same, as long as each can be derived from the other in a reasonable amount of time.

the asymmetric approach are retained, as is the speed of the symmetric system.

Another cryptographic tool is the *one-way hash function*, also known as a cryptographic checksum or a message fingerprint. A one-way hash function generates a relatively short hash value from a longer piece of cleartext. A typical hash size is on the order of 128 bits. Knowledge of the hash value does not provide any information that could be used to reconstruct the input text, or for that matter *any* cleartext that would hash to the same value. Furthermore, two nearly identical pieces of input cleartext hash to significantly different values. This means that one-way hashes can be used to verify that two large pieces of data are identical, without revealing any information about the nature of the data. Later, two applications of one-way hashes are described: digital signature and one-time passwords.

To be useful, ciphers and one-way hashes must be incorporated into a *cryptographic protocol*. This is a formalized dialogue between two or more parties that employs cryptography to enhance security. An example would be the procedure described earlier, by which two agents exchange a symmetric session key using a public key system. The exact syntax and semantics of the interaction are defined in a protocol. A strong cryptographic algorithm is rendered useless if it is used in a flawed protocol.

Finally, note that even an excellent protocol that employs a strong cipher can be rendered insecure by a poor implementation. A real-world example of this is presented later, but in the meantime consider an implementation of a secret key exchange protocol that retains the key in memory after it has been shared. An attacker could force a core dump and use a debugger to retrieve the key, regardless of the strength of the cipher or the cleverness of the communications protocol.

5.2/ Common Ciphers and Hashes

A detailed technical discussion of cryptographic algorithms is far beyond the scope of this paper. In this section some common contemporary ciphers are briefly discussed.

Symmetric Ciphers

The Data Encryption Standard, or *DES*, is the U.S. Government's "official" cipher[25]. (It was also adopted by ANSI, the American National Standards Institute, under the name DEA, where the 'A' stands for "algorithm".) DES is a 64-bit block cipher, meaning that it converts cleartext to ciphertext in 64-bit units. A DES key is 56-bits long, which provides a keyspace now considered to be too small.

One way to use DES more securely is to employ it repeatedly. *Triple-DES*, or *3DES*, uses three keys and three passes of the DES algorithm and is considered secure.

Skipjack is an algorithm designed by the National Security Agency. It uses an 80-bit key. Skipjack gained attention because it was to be the cipher used in the infamous *Clipper* chip, which became subject of intense controversy. Skipjack itself is a secret algorithm, which means it is not open to scrutiny by the general cryptography community. Algorithms that *rely* on being kept secret are fundamentally insecure. Although it is unlikely that Skipjack falls into this category, it is certainly not subject to the level of scrutiny to which publicly available algorithms like DES are subjected. Another concern was that Clipper's design deliberately included a "back door" that would enable a person knowing a particular key—a different key than the one that was used for encryption—to decrypt ciphertext generated using the chip. The U.S. Government planned to split these special keys and store them in two independent "escrow" databases. If an agency acquired a court ordered wiretap, the key halves would be retrieved and combined, and the target's supposedly secure communications would be vulnerable. Note that this is a "feature" of the Clipper chip, not of Skipjack itself (although the secrecy of the algorithm precludes confirmation that Skipjack does not have its own deliberate weaknesses). Neither the secrecy nor the key-escrow scheme were the real cause of alarm, which resulted from the Government's intent to mandate Clipper's use. Under the original scheme it would be illegal for a commercial wireless telephone manufacturer to use any encryption system other than Clipper. As of mid-1998, the debate is unresolved.

RC2 and *RC4* were developed at RSA Data Security, Inc. Both are variable key-length algorithms; RC2 is a 64-bit block cipher, and RC4 is a stream cipher. Both are roughly an order of magnitude faster than DES and about as strong with a 56-bit key. Both algorithms were originally restricted (RSA proprietary), hence subject to some criticism, although RSA has permitted external analysts to examine them for the purposes of verifying their security. RC2 is now described in an Internet RFC[26].

RC5 is a newer block cipher that supports a variable block size and a variable key size. It, too, is a product of RSA Data Security.

The International Data Encryption Algorithm, *IDEA*, is a 64-bit block cipher that uses a 128-bit key[27]. It is generally considered secure and is comparable in speed to DES. Unfortunately IDEA is patented in some countries, and where that is the case its commercial use is forbidden. Otherwise, it is an excellent cipher.

Many other block ciphers exist. Interested readers are referred to Bruce Schneier's canonical cryptography text[3].

Asymmetric Ciphers

The first public key system, publicized in 1976, is called the *Diffie-Hellman* algorithm (named after its designers)[28]. Diffie-Hellman is more properly a key agreement protocol. It allows two agents, who otherwise share no information, to establish a shared secret key over an insecure transmission channel. Diffie-Hellman is still used for key exchange but cannot be used to encrypt or decrypt messages.

Perhaps the best known public key algorithm is *RSA*, named after its designers (Rivest, Shamir, and Adleman)[29, 30]. RSA can be used for encryption or decryption and for authentication. For a public key cryptosystem, RSA is old: it was invented in 1977. Thus RSA has survived more than twenty years of intense scrutiny—unlike many other public key systems proposed since Diffie-Hellman. Combined with its versatility, RSA's apparent security makes it quite popular. As previously mentioned, RSA is slow: at least 100 times slower than DES, and an order of magnitude or two slower than that if both are implemented in hardware.

There are alternatives to RSA, but it is by far the most popular public key system.

One-way Hashes

The best known hash functions are the message-digest algorithms designed by Ron Rivest. All three generate a 128-bit hash code. The first, *MD2*, was invented in 1989[18]. There is a known vulnerability, but only if the algorithm is incompletely applied[19]. MD2 was designed and optimized for 8-bit machines.

In 1990 Rivest invented a faster hash, called *MD4*, that was optimized for 32-bit architectures[23]. In 1995, MD4 was "broken"—that is, a technique was discovered that generated *collisions* in less than a minute on a contemporary home computer[20]. A collision occurs when two pieces of input data hash to the same value. If a collision can be quickly generated, the hash code is no longer trustworthy as an indication that the original data is unmodified. As a consequence, MD4 should not be used.

Even before this vulnerability was found, Rivest redesigned MD4 to make it more secure. The result was *MD5*, which is slightly slower than MD4 but much more secure[24]. MD5 is the most commonly used one-way hash function.

The main alternative to MD5 is the Secure Hash Algorithm, *SHA*, developed by the U. S. Government's National Institute of Standards and Technology[21]. SHA is similar to MD5 but produces a 160-bit hash and is slightly slower. SHA had an unpublished flaw that was corrected in *SHA-1*, published in 1994[22]. SHA-1 has no known cryptographic weaknesses.

5.3/ Routine Encryption

One obvious solution to the problem of password interception is to stop transmitting cleartext passwords, and one straightforward way to do that is to encrypt the passwords before sending them. A broader solution that addresses privacy and the peripheral risks associated with network traffic interception is to encrypt *all* network traffic. This has an added benefit: if passwords are the only encrypted network traffic, it becomes much easier to sniff out the encrypted password and subject it to a cracking effort.

There are two serious objections to the adoption of encryption as a standard component of a network protocol stack—the net effect of routinely encrypting network traffic. The first is the cost. Executing encryption algorithms costs CPU time. An interactive login session, with a relatively slow thinking and typing user at one end, incurs very little CPU cost, although the encryption overhead may add latency. For high volume transmissions, such as file transfers to a mass storage system, the CPU cost of decryption may place a significant load on the target machine. Should this prove prohibitive, there are several options. The data can be transmitted unencrypted if they are not sensitive, or they can be encrypted at the source and then transmitted through an unencrypted channel.

The second objection to routine encryption is a practical one. The "standard" network utilities—that is, the ones that are shipped with most operating systems—generally do not support encryption. Fortunately, as is discussed in more detail later, there are a variety of low cost or free third party packages of excellent quality available to provide these services.

5.4/ Authentication

One way to use encryption to greatly improve security is to build a cryptographically strong authentication system. Both symmetric and asymmetric approaches can be used to build such a system. Throughout this discussion the Kerberos term *principal* denotes an agent with an identity that may be authenticated. An *authenticator* is a piece of data that asserts the principal's identity.

Private Key Authentication

In an authentication system based on a symmetric cipher, the ability to encrypt data using a particular key implicitly authenticates the key holder.

This introduces a paradox. In order to authenticate a key holder, one must be able to decrypt a proffered authenticator. But to do that, one must hold a copy of the secret key. Since possession of a key implies identity, sharing the key implies sharing identity. Distributing the key to all those who would want to verify the owner's identity would obviously defeat the point: all of the people with the key would now be capable of masquerading as the key owner.

The solution employed by systems like Kerberos is to create a trusted third party that stores the keys of all principals. This effectively bootstraps the system. Consider a simplified scenario in which agent A wants to authenticate to agent B. (This is purely explanatory and not intended to describe the behavior of any particular real-world authentication system. In fact it is flawed, but illustrates the general approach.) First, A sends a message encrypted with its secret key (A) to the central authority. The central authority decrypts the message using key A: if the decryption produces meaningful data, the person who sent it must be agent A. The central authority can then encrypt a message verifying the identity of agent A, using agent B's key (B). Nobody but agent B can decrypt the message, so it cannot be meaningfully tampered with. The central authority returns the new message to agent A, which sends it to agent B as proof of its identity. If the message is legitimate, B can decrypt it. Since it knows that the message could only have been created by the central authority, which is implicitly trusted, and it knows that the central authority would have delivered the message to A only if it could verify A's identity, B can be confident that A is who it claims to be. Note that the scheme has two major dependencies: keys must never be shared, and the central authority must be absolutely trustworthy.

In a real symmetric encryption based authentication scheme, such as Kerberos, a number of complications are introduced. For example, steps must be taken to ensure that authentication messages can not be intercepted and "replayed" by attackers, which would allow them to adopt the identity of the legitimate recipient (even if they are unable to decrypt the message to view the original authenticator). There is also an operational requirement to reduce the number of times a user has to type in a password: once for every authenticator would be far too inconvenient.

Symmetric encryption can be used to build a strong authentication infrastructure. Its major disadvantage is the requirement for the installation and management of a central authority. The central authority must be be trusted absolutely by all parties wishing to use the infrastructure: obviously not a solution that scales well. The central authority also becomes a critical resource that must be protected at virtually all costs. And it is a single point of failure, unless additional complexity is introduced to replicate the authority's services (which will also increase risk, since there will now be multiple private key archives).

PUBLIC KEY AUTHENTICATION

An authentication system based on asymmetric cryptography does not require an absolutely trusted third party key archive, so it avoids several of the drawbacks of private key systems.

Again, consider a case where principal A wants to authenticate to principal B. (And again, this simplified example is purely illustrative.) The first step, usually completed before the beginning of the authentication protocol, is for A to distribute its public key. (The issue is addressed in more detail later, because it turns out to be the trickiest part of asymmetric authentication schemes, but for the moment assume that B has A's public key and is absolutely certain that the key belongs to A.) Once this precondition is met, authentication is simple. A encrypts a message using its private key and passes it to B. If the public key decrypts the message then it came from agent A. Note that knowledge of the public half of the key does not allow B to masquerade as A, because messages encrypted using the public key cannot be decrypted with the public key. This means that the public key, and hence the ability to authenticate the private key holder, can be freely distributed without compromising the the authentication system.

With an authentication system based on an asymmetric cipher there is no requirement for an absolutely trusted third party key archive, but there are still trust issues. How does agent B know that the public key it holds really belongs to agent A, and not some third party advertising itself as A? An attacker could simply pretend to be A, generate a new key pair, and hand B the new public key. As long as B did not already have a copy of A's real public key, the substitution would go unnoticed.

The problem of reliably associating principals with their public keys is mitigated by some useful characteristics of asymmetric cryptosystems. The first is that an association only has to be established once (unless the private key is compromised and the principal is forced to generate a new key pair). The next is that because of the nature of asymmetric cryptosystems the public key can be freely disseminated. This means that a principal anticipating involvement in an authentication network can distribute its public key to a number of key servers. Those wishing to discover the principal's public key check with several servers and raise an alarm if the returned keys are not identical. This means that a potential attacker must compromise all copies of the key mapping. In other words, by more greatly disseminating the public key one actually increases its security.

There are more complicated key registration schemes that rely on digital signatures. If A knows B personally, he or she can personally vouch for the key mapping by "signing" a certificate that asserts the truth of B's claim to its key. When B distributes its key it also distributes A's certificate of authenticity. If A is well known or if

B has enough certificates from enough principals, the recipient can feel confident in accepting the key. (As shown later, digital signatures can cryptographically guarantee the integrity and origin of a piece of data, such as a copy of a public key.)

Despite the existence of these schemes, management of the *(principal, key)* mapping is the most difficult aspect of authentications systems built on asymmetric ciphers.

Host Authentication

One way to make spoofing attacks more difficult is to assign keys to hosts as well as users. In other words not only can people be principals, but computers can be too. For that matter, pieces of software, such as well known servers, can be authenticate-able principals. A host authentication infrastructure works analogously to the user authentication schemes described above. Symmetric or asymmetric cryptosystems can be used.

A simplistic public key approach would be for each host to store its private key in a file readable only by root but to advertise the public key as much as possible. When initiating a connection, a short cleartext message—the authenticator—is encrypted with the private key and delivered to the target machine. The public key is used to decrypt the authenticator, and if the resulting text is meaningful the originating host's identity is verified.

Once a cryptographically strong host authentication system is installed, spoofing attacks become much more difficult. An attacker needs knowledge of a machine's private key. (The attacker's alternative would be to replace all the copies of the public key with a spoofed version, and once again, good distribution of the public key makes this extremely difficult.) If the host's private key has been properly protected, the only way to learn it is by learning the host's root password.

Note that as with all other cryptographic systems, the protocol and its implementation must be secure. This can be difficult when dealing with non-human principals, because they have no intrinsic knowledge that can be used to protect the private key. A human can encrypt the private key using a good pass-phrase, so that it can not be stolen. A computer or a piece of software simply has to store the private key on disk and protect it using the filesystem's access control mechanism. This introduces a wide array of potential vulnerabilities.

Despite these difficulties, it is fairly easy to construct a reasonably secure host authentication infrastructure, and doing so provides a profound improvement in security by making host spoofing substantially more difficult.

5.5/ Two Factor Authentication

One way to improve even a strong authentication system is to require more than just a password for proof of identity. This is known as two factor authentication, and has the advantage that compromise of a password does not result in a complete break-down of the authentication system. There are two basic schemes: one is to require the use of a physical device, for example a card that generates unguessable but verifiable identification codes, so that ownership of something is required in addition to knowledge of a password. The second is to actually sample a physical feature of the user, for example a fingerprint. The former is simpler and currently cheaper; the latter is more secure and avoids the problem of lost or forgotten authenticators. For the sake of this discussion both schemes are referred to as "physical token" authentication.

These physical token schemes are complementary with the encryption-based schemes described above. Both are useful on their own, but are much more powerful when combined. The physical schemes close a hole that encryption does not address: interception of data as it is typed on a local machine, before it is encrypted. On the other hand they add overhead and can be inconvenient. One compromise is to require physical tokens only for connections originating outside the local network.

5.6/ One Time Passwords

One time passwords (OTPs) are about as close to cryptographically unbreakable password security as can be achieved. The idea is that a password is valid only once, so that intercepting it as it is used is useless. This is implemented using a one way hash algorithm, for example MD5. As described earlier, one way hash algorithms compute an output code based on an input pattern in such a way that effectively no information about the input pattern is retained in the output—in other words there is no way to go "backwards" from the output to the input.

Here is a typical implementation of a one time password system. Prior to any use of the system for authentication, there is an initialization phase. The user is prompted for a secret password; the system reads it and repeatedly applies the one-way hash, say for the sake of this example 100 times. The result is stored in a local file. There is no way to recover the password from the stored hash value, because of the one-way property of the hash algorithm.

Assume that after initialization, our user goes to a remote site and wants to authenticate. After running a login program, he or she receives a challenge that includes the current hash count (100 in this case). He or she runs a local program, which can be carried on a floppy disk, a palm- or lap-top computer, or even in a hardware device such as a credit-card sized encoder. The program reads the password and the hash count (n) and generates the n-1'th hash code—in our example the hash code resulting from running the algorithm 99 times. The user then responds to the challenge by typing (or cutting and pasting) the 99-hashed password as a challenge response. The target computer knows only the 100th password in the sequence, but by taking the 99th password as delivered by the user and hashing it one more time it can generate the 100th hash. If the newly generated code matches the one recorded in the local file, access is granted and the 100th hash is replaced with the 99th hash just provided by the user. The next time that the user wants to authenticate, the key is hashed only 98 times, but otherwise the same procedure is used.

If the user will not have access to a secure platform on which to generate one time passwords, he or she can pregenerate a list and physically carry it in a wallet or purse. In our example, the user might ask for the 90th to the 99th passwords. There is a risk of losing the password sheet, but because each password can only be used once—and all of them can be easily discarded by setting the server's hash count to the next number below the lowest numbered password that was lost—the cost of such a compromise is not too severe.

One time password schemes usually map the otherwise cryptic hash code to a sequence of English words. These are not meaningful except that each of them is, by convention, mapped to a certain bit pattern. When these patterns are catenated they form a valid hash code representing N iterations of the hash algorithm with the user's pass-phrase as input.

5.7/ Digital Signature

Digital signatures provide a way for the originator of a piece of data to cryptographically guarantee its integrity and source. The goal of a digital signature system is to accomplish through strong cryptography the same guarantees normally associated with "real" signatures. (In fact, digital signatures provide superior integrity and authenticity guarantees.)

Although in theory a symmetric cryptosystem can be used for digital signatures, in practice a public key system is much more useful. A symmetric system requires a third party trusted by both signer and recipient of the signed document. This is unwieldy for the same reasons that a symmetric authentication systems are unwieldy, and because of the way signatures are used it is often less practical. All contemporary digital signature schemes use asymmetric cryptosystems.

An alert reader will no doubt realize that once a public key authentication system is in place there is no real need for digital signatures. The same goals can be satisfied by generating the original message, encrypting it with the author's private key, and distributing the public key. Recipients of the encrypted message know that if they can decrypt it using the author's public key it must have been encrypted with the private key, hence it was created by the author. They can also be sure that it has not been tampered with, because there is no way to do so in such a way that the decryption would then produce meaningful data.

Digital signatures offer additional features that make them more useful that basic asymmetric encryption. First, recall that asymmetric ciphers are typically computationally expensive. The cost of encrypting entire messages may be prohibitive. Second, consider a situation where a document must be signed by several principals. To do so using standard encryption requires the generation of multiple encrypted copies of the document, each at least the size of the original.

To address these problems, digital signature schemes usually employ a one-way hash algorithm to generate a "fingerprint" of the original document. This fingerprint is then encrypted using the signer's private key. Because of the properties of one-way hash algorithms, it is extremely unlikely that any other document would hash to the same fingerprint (on the order of 1 in 2^{128} for MD5, for instance). Furthermore the fingerprint will be short (128 bits for MD5) hence can be quickly encrypted using an asymmetric key, even if the cipher is very expensive. Multiple signatures can be appended to a document without significantly increasing its size.

5.8/ SUMMARY

Cryptographic techniques can be used for privacy, authentication, and digital signature. Routine network traffic encryption reduces the amount of information available to attackers, which makes their task more difficult. The combination of strong user and host authentication can make spoofing very difficult. Two-factor authentication practically eliminates the damage caused by a stolen password, and one-time passwords can be safely used from a host known to be compromised. Digital signatures provide an inexpensive, cryptographically strong mechanism for guaranteeing the origin and integrity of a document. The combination of these techniques effectively eliminates some of the most glaring vulnerabilities in contemporary computer security systems. The next question is whether implementations of this technology are available, and answer is that there are a number of well designed tools to choose from. Some are described in the next section.

6/ DEFENSIVE TOOLS

In order for the cryptographic techniques described in the last section to be useful, they must be integrated into practical tools. Here some of the most useful contemporary security tools are described.

6.1/ KERBEROS

Kerberos is an authentication system developed at MIT and based on symmetric (private key) cryptography. Its primary purpose is to provide strong authentication across an insecure local area network[5, 6].

Kerberos works very similarly to the generic private key-based authentication system discussed in the "abstract solutions" section. As with all symmetric cryptography systems, a central, absolutely trusted key manager is required. This is known as the Key Distribution Center, or KDC. Kerberos is not limited to authenticating users: hosts or network servers can also have cryptographically verified identities. The general Kerberos term for an authenticated entity is *principal*.

Authentication information is stored and transported in encrypted packets of information called *tickets*. To use a particular network service, a principal asks the KDC for a ticket for that service and then uses this *service ticket* to authenticate to the target server.

The first step in using Kerberos is acquiring a ticket for the key distribution center itself. This special ticket is called the *ticket granting ticket*, or

TGT. The TGT is a data structure that holds the principal's identity, the time the TGT was generated, the machine to which it was delivered, and some Kerberos-specific flags. It is encrypted using the KDC's secret key, so that an attacker–including the principal that acquired the TGT–cannot meaningfully tamper with it. Because of this property a TGT, or a Kerberos ticket in general, is said to be *opaque* (i.e. it is not possible to "see" into the data structure).

For human principals (i.e. users), the TGT is acquired using an initialization program that prompts for a password. Note that the password is not passed over the network; instead, before the user enters it, the KDC is asked for a TGT encrypted with the user's password. This opaque ticket is passed back to the initialization program. The user is then prompted for a password, which is used to decrypt the newly received TGT[3]. Successful decryption, i.e. the production of a correctly formatted Kerberos TGT, implies that the user knows the correct password and thus validates his or her identity. Once the TGT has been acquired, it is stashed in a credentials cache on the principal's local machine.

When the principal wants to use a network service that participates in the Kerberos authentication infrastructure, it has to acquire a *service ticket*. This is another data structure similar to the TGT structure, this time encrypted using the secret key of the target network service. To acquire the service ticket, the user sends a message to the KDC. The message includes the TGT, the service name for which a ticket is required, and some other information. (The entire ticket request is encrypted with a one-time session key established during TGT acquisition. In fact, almost all communications between the client principal and Kerberos-equipped servers are encrypted using a session key negotiated the first time the client and server interact.) When the KDC receives the request package tries to decrypt the TGT and verify the user's identity. If this succeeds, the TGT generates a new package of authentication data and encrypts it using the secret key of the network service. This ticket is opaque to the user, but can be decrypted by the network service. It is returned to the requesting principal to be used when convenient (although tickets generally have a fixed lifetime, after which they become invalid).

Note that the KDC is a central player in the authentication system, involved in almost all transactions. In addition it is implicitly and absolutely trusted. Thus good administration of the KDC is essential for the smooth and secure operation of a Kerberos *realm* (the set of machines over which a particular instance of Kerberos operates, often roughly analogous to an Internet domain in size).

In order to take advantage of Kerberos's authentication system, programs have to use a Kerberos API (application programmers' interface). This means that "legacy" applications–those that were not written specifically to use Kerberos–have to be modified before they can take advantage of the authentication infrastructure. The Kerberos distribution includes versions of the most common user-invoked network programs, including *rsh*, *rcp*, *ftp*, and *telnet*, for example. There is also a version of *su* and a number of Kerberos-specific administrative utilities.

In addition to authentication services, Kerberos optionally supports the encryption of network data streams. The rewritten applications included with the distribution also optionally use this service, so that user login sessions or data transmissions can be encrypted.

Kerberos is a mature, freely available software package that provides proven strong authentication. Users, machines, and programs can all be authenticated with confidence, reducing the risks of spoofing attacks. Kerberos never passes a clear text password over the network, and it passes an encrypted password only when absolutely necessary—when setting or changing one. The system is designed to avoid requiring the user to type his or her password more than once a day (or so—the actual time period is configurable). Once the password is used to acquire a TGT, all

[3]In the case of a machine or server principal, the password is stored in a key table file, called a *keytab*. This is a potential security risk, since failure to properly control of access to the keytab can compromise the key.

other Kerberos activities happen automatically, without user intervention—except, of course, that the user is required to run the Kerberos-capable versions of network utilities.

The Kerberos documentation is reasonably well written and complete, and on-line FAQs and other helpful information is easily available. The software is distributed as source code and compiles on a wide variety of UNIX platforms. Macintosh is also supported, as well as a 16-bit Microsoft Windows port. A Windows NT port is in progress. The administrative tools and Kerberos-enabled utilities that are included with the base system are generally functional and have few bugs.

In short, Kerberos is a stable, highly functional system that works on most platforms. It is well understood and provides a good authentication infrastructure, with the added bonus of supporting encrypted data transmission.

Unfortunately, the implementation of Kerberos does have some weaknesses[7]. In particular, a person with root privileges on a Kerberos host can cause a number of problems. First and foremost, credentials cache files, which store the TGT and all the tickets it has helped acquire, can be read and stolen by a root-privileged process. This means that root can potentially masquerade as any other principal, once that principal has initially authenticated. Of course, the limited lifetimes of Kerberos tickets limit the damage that can be caused (and credential caches can be destroyed at will or automatically at logout, which helps). Root privileges also give access to all secret keys stored on the host, which includes the host key itself.

These problems are potentially serious, but one should consider that a rogue root user can make many other potentially more damaging attacks. For example, a root user can snoop the kernel clist data structures to steal passwords as they are typed, which is potentially far more serious than the temporary masquerade provided by credential stealing.

There are some practical issues, as well. Kerberos is not a "light-weight" solution. It requires that each host in the realm be equipped with a variety of Kerberos daemons and client applications, as well as at least two configuration files. A substantial overhead in administration is required, and of course there are the reliability and security issues associated with having a single point of failure—the KDC. There are provisions for configuring slave KDCs, which can operate when the primary is off line, but these require administrative overhead and are additional vulnerabilities comparable in risk to the master KDC. There are also the problems of providing physical security for the KDC and its slaves.

There is the issue of supporting "legacy" applications—that is, the requirement that programs be rewritten to be able to take advantage of Kerberos' authentication system. There are also some common scenarios in which Kerberos is unable to protect against clear text password interception. For example, a common idiom is to start an *xterm* on a remote host using *rsh*, displaying the results locally. Although Kerberos can make the initial connection, it is no longer involved in the communication path between the local *X* server and the remote client—all traffic, including any passwords that are typed, are now exchanged as clear text.

Kerberos requires substantial education and retraining of the user community. The concepts employed are not particularly difficult, but the first time they are encountered they can be daunting. There are new commands for users to learn, and the Kerberos implementations of the standard network clients behave somewhat differently from the usual versions, in some cases taking different options—or worse, using the same options to enable or disable different features.

Kerberos is not tremendously difficult to acquire, build, and install, but the task is substantial enough to prevent casual users from undertaking it. In addition, for it to be properly installed one must have root privileges. This means that users who are connecting from outside sites will be able to use Kerberos only if their administrative staff have installed it already, or if they themselves are sufficiently skilled and motivated to install it.

6.2/ Secure Shell

Secure shell, or *ssh* as it is more commonly known, is a package of software that uses asymmetric cryptography to provide encryption services and greatly improved authentication[8, 9, 10, 11]. It is a smaller package than Kerberos, both conceptually and in terms of required computational and administrative resources. Furthermore it takes a fundamentally different view than Kerberos, in that it is designed to be deployed and used by individual users, rather than by a central organization that has control over all the machines operating on a LAN. This approach has its advantages and some disadvantages.

Used in its most simple mode, ssh is a drop-in replacement for *rsh*, *rlogin*, and *rcp* that provides strong encryption of the data stream. A variety of encryption algorithms are implemented and can be chosen via command line options or a per-user or per-site configuration file. *Ssh* can either replace or coexist with the original network applications.

Basic encryption services are useful on their own, but *ssh* provides additional features. The first is an improved authentication procedure. By choosing options appropriately, users can select one of several levels of stronger authentication protocols. The most simple and least secure is to use BSD `ruserok()` just as *rlogin* does. In this mode the only advantage provided is data stream encryption. In cases where `ruserok()` requires a password—e.g. if there is no entry for the incoming host in the *hosts.equiv* or per-user *.rhosts* file—*ssh* encrypts the password before transmitting it to the server.

The next level of security is to add RSA-based host authentication. In this mode, each host is assigned an RSA (asymmetric encryption) key pair. The public key is freely advertised; the private key is kept in a file local to the host and readable only by root. (The private key is generated automatically when *ssh* is installed.) When connecting to a host for the first time, *ssh* asks the user if he or she would like to download and save the target host's public key. The target host then transmits an authenticator encrypted in the private "half" of the key; if the client side can decrypt the authenticator, access is permitted. Now that the public key associated with the host has been saved (either in a user-specific directory or in a host-wide table), future connections will be permitted only if the advertised public key matches the saved one and the authenticator can be decrypted. If the advertised public key is different from the saved one, the user is warned that a man-in-the-middle or spoofing attack may be taking place. Optionally, having determined that no such attack is occurring, the user can replace the old public key with the new one. Correctly used RSA-based host authentication effectively detects and prevents IP, DNS, and routing spoofing attacks—assuming of course that the user running *ssh* pays attention to the alarm messages and understands what they mean.

The highest level of security provided by *ssh* supplements host authentication by assigning users an RSA key pair. The private key is stored on the user's local machine, encrypted using a pass-phrase. The user's public key is stored on the server side. When a connection is attempted, the server challenges the client with a random number encrypted using the public key. If the client has the private key, it can decrypt the challenge and tell the server what the random number was. This proves that the client owns the key pair, and access is permitted if the public half of that pair is listed in a per-user file of authorized keys.

One complication here is that in order to access the private key, which is stored encrypted on the local disk, a pass-phrase is required. Hence every time a connection is made, the user is required to type a passphrase. Fortunately the *ssh* package includes another program called an authentication agent. This is a background process that starts when the user logs in on his or her local machine. It prompts for the pass-phrase and uses it to decrypt and load the private key. *Ssh* is designed to check for the existence of an authentication agent when a login connection is being made. If there is one active, it can employ the private key on behalf of the user it represents, without having to prompt him or her.

An *ssh* server can be configured to accept connections at any of the security levels described above. For example, if the encryption services are all that is required, *sshd* can be configured to accept any connection that *rshd* would accept.

In addition to encryption and superior authentication services, *ssh* provides a tunneling facility that enables it to forward arbitrary TCP/IP connections through an *ssh* connection, so that they are encrypted without any change to the legacy application being required. *Ssh* includes some special facilities to make this particularly easy with *X11* connections. A proxy *X* server is established on the remote side; all *X* clients started by the user on that side are directed to use the proxy as the server. The proxy uses the tunneling mechanism to forward *X11* protocol messages back through an encrypted channel to the user's local *X* server.

Ssh can be dropped in as a nearly transparent replacement for existing applications[4]. As such, it provides no extra authentication support but does provide full encryption services. This alone is a valuable feature. As users become more comfortable with the new tool, they can gradually increase the level of security they employ.

Furthermore, the client version of *ssh* can be easily installed by a reasonably knowledgeable user, without root privileges and without requiring control of the client machine. This means that users at remote sites can be plausibly expected to install the software.

In general, *ssh* can be installed and used without an organization-wide commitment such as is needed for Kerberos. Furthermore, no central administration is required, other than the initial installation.

6.3/ S/KEY AND OPIE

S/Key[15] and *OPIE*[16] are software implementations of one-time password schemes. Both are based on one-way hash functions, described earlier. *S/Key* uses MD4, which has been broken, hence *OPIE* is preferred.

By their nature, one time passwords are useless if intercepted. This makes them ideal for use from untrusted systems and across untrusted networks. In addition, because a list of OTP's can be prepared ahead of time and carried with a traveling user, they can be used to provide secure access from any machine.

One disadvantage is that both *S/Key* and *OPIE* either require a local client program that generates the OTP based on the user's password and the server challenge, or a pre-generated list of OTPs that the user carries with him or her. In the latter case, there is a risk that the user will lose the password sheet. Still, these tools allow secure access from untrusted remote hosts over untrusted networks. If users would be making the connections in any case, using insecure tools like *rlogin* or *telnet*, the risk of a lost OTP sheet is relatively small.

6.4/ SECURID TOKENS

A company called Security Dynamics Technologies manufactures SecurID authentication tokens. These are essentially credit-card sized hardware implementations of one time password schemes. Several alternatives are offered[31, 32].

The most simple is a card ("token") that simple generates a unique access code every sixty seconds. The sequence of codes is unpredictable. To login, a user types a personal identification (PIN) code and the current access code displayed by the token. The remote server maps the PIN to the cryptographic parameters associated with that particular token, and by doing so can verify that the provided access code is being generated by the correct token.

In some cases there may be concern that the PIN can be intercepted. This could lead to a security breach, either because the intercepted PIN and access code could be reused within the sixty second lifetime of the code—unlikely—or because

[4]One area where *ssh* falls short of true transparency is as a replacement for *rcp*. *Scp* has similar functionality, but unlike *ssh* it requires *sshd* on the remote end: there is no automatic fallback to the *rcp* protocol.

the access token itself could be physically stolen and used with the PIN. For customers concerned about this type of problem, Security Dynamics sells a token that incorporates a keypad, so that the user can enter the PIN and generate an access code that combines it with the standard time-sensitive access code. The combined code is then used to respond to the remote challenge, and if everything matches up access is permitted.

The great strength of the SecurID scheme is that it introduces a second authentication factor. This dramatically improves security, making interception of passwords only a partial compromise. If nothing else, this buys time for security administrators and gives them the opportunity to change passwords, revoke authentication tokens, or take whatever other emergency measures are appropriate.

SecurID also permits login over untrusted networks and from untrusted hosts, in the same way that software OTP schemes do. And the physical token approach has the advantage of not requiring a local program for generating the OTP from the PIN, or a dangerous list of pre-generated OTP's.

The main problem with two factor authentication schemes is the same as its strength—the requirement for possession of a physical authentication token. This introduces the possibility of losing or forgetting the token, and of course the added complexity of purchasing and distributing the tokens.

SecurID also requires a server product on the remote side, which ties into a database that maps PINs to card serial numbers for determination of the correct parameters to the encryption routines. This entails a financial commitment to Security Dynamics, as well as introducing training and maintenance issues.

6.5/ PGP

PGP stands for "Pretty Good Privacy". PGP is an asymmetric cryptography-based email authentication scheme that also provides digital signature and encryption services[33, 34]. PGP allows users to generate asymmetric key pairs for themselves and then to manage and use these key pairs. Public keys are managed using virtual key rings, which record the entity associated with the key, the key itself, and the identity of other users who have signed the key to verify that they believe that it really belongs to the person who claims it. PGP allows association of trust ratings with signators, so that a trust value can be calculated for each entry on the key ring.

Although PGP uses a public key system for authentication, bulk data are encrypted using a symmetric cipher. The key for the symmetric cipher is chosen randomly at the time the message is encrypted. It is passed to the intended recipients as part of the message but encrypted using their public keys. The recipients' software extracts the encrypted key, decrypts it using the appropriate private key, and then decrypts the message contents using the resulting symmetric key. Thus a single encrypted message can be decrypted by multiple recipients, without requiring a shared secret key and without requiring that the message be replicated several times. An added benefit is that the symmetric cipher is usually much more computationally efficient, hence the cost of encrypting and decrypting is kept low.

Digital signatures are generated by calculating a checksum for the message and then encrypting it using the author's private key. Recipients know that the message originated from the author, and has not been tampered with, if they can decrypt the signature using the author's public key and the decrypted checksum still matches the message.

Encryption and signature can be combined, so that the intended recipients are the only people who can read the message, and in addition they can verify that it originated from the claimed author and has not been altered.

There are several mail tools that understand and automate the process of encrypting, decrypting, signing, and verifying messages using the PGP software. (For example, *PGPmail*, or various *Emacs* packages.) This makes use of the system considerable simpler for most users.

PGP makes email secure: a recipient can be confident that a message comes from the purported author and has not been altered in any way. If encryption is used, the recipient also knows that nobody else has seen the message in transit. This means, for example, that PGP can be used to email out a set of one time passwords for somebody at a remote site. The digital signature features provide assurances that allow email to be used as a first class component of a bureaucracy. For example, if a user requires a root password for a certain machine, she can send a signed message to the machine's administrator asking for the password. The administrator knows that message comes from the user and can then send a signed message authorizing distribution of the password to the security managers. They can then verify the administrator's identity, and the user's, encrypt the root password using the user's public key, and email it to her.

PGP is fairly complicated. A significant amount of background reading and studying of the documentation is necessary before the system can be used securely. Like all security software, PGP can be misused, which introduces the nasty possibility of people emailing sensitive information under the delusion that it is safe from prying eyes, only to find that it is in fact exposed.

PGP is probably overkill for routine messages (although an argument for routine digital signing of messages can certainly be made).

6.6/ Secure Socket Layer

The Secure Socket Layer (SSL) is an Internet protocol that provides authentication and encryption services[35, 36]. Authentication is implemented using asymmetric cryptography, and encryption using a symmetric session key negotiated at connection initiation. SSL performs authentication and encryption prior to any user data transmission.

SSL is comparable to Kerberos in some ways, in that applications must be modified to work within its authentication infrastructure. Unlike Kerberos, however, the emphasis with SSL has been on codifying an infrastructure that can be used to develop custom secure applications, for example for use on the Web. SSL does not include any description of key management techniques, specific applications, or tools for managing a system built using the basic protocol. This approach contrasts with that of Kerberos, which concentrates on a particular implementation of an authentication system, the tools for managing it, and standard general purpose applications that use it.

As a programmer's tool, SSL is an interesting technology that may become increasingly useful as authentication systems and applications are built around it. However, SSL has no immediate utility as a component in a general purpose computing environment, primarily because there are not enough appropriate applications and tools implemented.

6.7/ DCE

The Distributed Computing Environment (DCE) is a comprehensive infrastructure for distributed computing produced by the Open Group[37]. The security component of DCE is an older version of Kerberos, slightly modified to work within the DCE environment[38].

DCE carries all of the advantages and disadvantages of Kerberos. Additionally it has the advantage of providing services other than security that can be used for distributed computing.

Unfortunately DCE also introduces problems that are not shared by Kerberos. Perhaps first among these is a requirement for a site-wide commitment to deploying DCE, which requires purchasing the DCE software for all architectures. In some cases DCE may not even be available. Furthermore, the DCE source code is expensive and the system is so large that porting it is impractical. The complexity of DCE and the difficulty of integrating new versions of its component software leads to it lagging behind the state of the art. For instance the version of Kerberos incorporated into the DCE Security Service is an older one that lacks several important facilities. For the types of problems addressed in this report, DCE offers similar features to Kerberos but carries more disadvantages. For this reason it is not considered further.

6.8/ FIREWALLS AND PACKET FILTERS

A *firewall* is an Internet router that mediates all communication between internal networks and the Internet as a whole. The firewall host (or a few hosts, although for convenience it is assumed there is only one) is the only direct point of contact from the internal network to the outside world. All traffic to and from the Internet flows through the firewall.

A firewall can greatly decrease the vulnerability of internal hosts to external attack. Since there is a single entry point to the internal network, a security administrator can spend a great deal of effort to keep it secure. This is a typical solution if resources are too scarce to secure each individual machine on the internal network. Firewalls can also result from an overly centralized approach to security, in which the general user community is not trusted or the security administrator refuses to relinquish any responsibility (and hence authority) for security.

The dangers of this approach are three-fold. The least important relates to performance and reliability. A firewall is a natural bottleneck and single point of failure. This is mitigated if there are multiple firewall hosts, but they are still obvious candidates for an organized denial-of-service attack. Secondly and more seriously, the firewall is an obvious focus for attackers, a virtual bullseye, which leads to the third and most serious problem. Once an attacker has penetrated the firewall, either by cracking it or because he or she already has legitimate access to the internal network, the internal hosts are open to attack.

This scenario is particularly dangerous if the internal hosts have been poorly administered (from a security point of view). In many cases this is part of the motivation for establishing the firewall in the first place. If the internal machines are adequately protected and monitored, a firewall is redundant.

Although firewalls in the traditional sense may cause as much harm as good, a slight modification of the scheme can be very helpful. If access gateways are equipped with flexible packet filters, they can help to reduce the risk of certain spoofing and denial of service attacks. For example, gateways can be configured to block external packets with originating addresses that belong to internal hosts: in other words, packets generated by an outside attacker in an attempt at spoofing an internal host. Also the ability to "block" a particular Internet host or domain can be useful in emergency or nuisance situations.

Clearly a packet-filter type of approach can be useful. The key is to continue to attend to the internal hosts, so that if the virtual perimeter is compromised they are individually protected.

In short, the firewall (packet filter) approach is useful if it supplements the other recommendations and tools given here; not as a replacement. The other useful scenario occurs when the internal machines are *incapable* of protecting themselves. Some commercial operating systems are sufficiently poorly designed that they simply cannot be made secure while remaining fully functional. In this case a strong firewall may be the only recourse.

6.9/ IPSEC: SECURE IP

IPsec—secure IP—implements SSH- and SSL-like authentication[13] and encryption[14] at the IP packet layer, i.e. low in the network stack. Ultimately, it is likely to replace the similar higher level schemes. However IPsec is currently not widely implemented, and again, as with SSL, there is a lack of ancillary software for managing systems built with it.

For the moment, IPsec is not useful for the majority of computing sites. When operating systems start to provide IPsec-capable drivers and IPsec administrative tools, it is likely to replace application-level approaches.

7/ Recommendations

Recommendation 1
Involve users.

This is critical no matter which of the subsequent recommendations are adopted. Users can short-circuit almost any security procedure, rendering it useless or worse than useless, unless security administrators are able to win their respect and cooperation. A security system is only as strong as its weakest link, and unless users are cooperating they will usually be that weak link.

There is, of course, no magical formula for building a cooperative relationship between security administrator and user community. Techniques that would work well under one administrator in a certain environment might very well fail elsewhere, depending on the requirements of the users and the personality and skills of the administrator. Nonetheless, there are a few general points that apply in most situations.

- Educate users. They will be much more likely to adopt and appreciate secure techniques if they genuinely believe that a threat exists. Web sites and short seminars are excellent ways to do this.
- On the other hand, do not exaggerate the threat. Security incidents really happen. Howard calculates a frequency on the order of one incident per Internet domain per year, and for big sites this number is probably higher. But few incidents cause any significant damage or problems. Misrepresenting the threat as dire and immediate will quickly become tiresome, and the administrator will lose credibility.
- Make security tools easy to use. One way to do this is to provide pre-built tool packages for the site's computing platforms. Along with documentation, these can be distributed from an internal web site. Another option is to provide floppy disks with one-time password clients or ssh clients, so that users visiting remote sites or logging in from home can easily use these tools.
- Remember that the security administrator is part of the system administration team, and that the system administration team exists to help the community effectively use the computer systems. Take this to heart, and try to convey this attitude to the user community. Ideally they will see the administrator as a person helping them deal with the unfortunate realities of a potentially hostile Internet environment, rather than an autocrat interfering with their work by introducing pointless obstacles.
- Avoid heavy-handed and authoritarian approaches wherever possible. Although there may occasionally be an extreme circumstance where an authoritarian approach is appropriate, this should be the exception rather than the rule. Authoritarianism breeds resentment and in some cases active opposition.

To summarize: the administrator should strive to educate users about the problems that exist, make it as easy as possible for them to use the tools that address the problems, and underneath it all try to position him- or herself as the users' ally.

Recommendation 2
Encrypt interactive login sessions.

Encrypted login sessions eliminate password-stealing sniffer attacks, which are a significant threat. There are negligible performance costs, far outweighed by the benefits. Either Kerberos or SSH can meet this requirement.

Recommendation 3
Introduce a strong authentication infrastructure.

As discussed earlier, the basic UNIX authentication system introduces host spoofing vulnerabilities which can lead to unauthorized access or password stealing. Host spoofing can be made much more difficult with the introduction of a cryptographically strong host authentication system. An improved user authentication system can support convenient no-password hops between machines without a dangerous system, like BSD's *rhost* approach. Since *rhost* vulnerabilities result in a significant number of compromises, *.rhost* and *hosts.equiv* files should be forbidden or strictly controlled. Either Kerberos or SSH can meet all of these requirements.

Recommendation 4
Provide one-time password access for connections from external sources.

If a user types a password over an unencrypted connection even once, that password is compromised. It is important to provide an access mechanism that can be used when the normal encrypted login tool is not available, and OTPs satisfy that requirement. S/Key meets this requirement but uses the broken MD4 one-time hashing algorithm; OPIE uses MD5 and is preferable.

Recommendation 5
Use two-factor authentication for initial login.

Of the recommendations, this is the most difficult to implement. However it greatly improves security by reducing the absolute dependence on password secrecy. Two-factor authentication should be required for access to the local network, whether from an external source or at a local user's point of entry. A less secure alternative is to require two-factor authentication for root login, but not for other users.

Recommendation 6
Do not permit group accounts.

Group accounts are dangerous and they are rarely necessary. They are a natural target for attackers and are much more difficult to effectively monitor, so that a break-in is likely to remain undetected for longer. If at all possible, group accounts should be avoided.

One alternative is to create individual accounts for each who would have used the group account. This may not be an improvement if the accounts are infrequently used, since they then become a target. In this case, the ideal is to dynamically create and delete accounts as needed. Another possibility is to use *Kerberos*, which can be configured to grant realm-wide accounts.

The one account that will almost certainly have to be shared is root. In many environments there is no practical way to avoid this. A shared root account is less risky if direct logins are not permitted. Direct logins are vulnerable to spoofing attacks and suffer from a lack of accountability, which makes post-incident investigation much more difficult. Users should first login as "themselves" and then become root.

Recommendation 7
Test system security by simulating attacks.

Monitor CERT/CC reports to keep track of the state-of-the-art in cracking attacks, and identify vulnerabilities in your system's security by using the same methods as attackers would. Run cracking programs to identify vulnerable passwords. Toolkits like *ISS* and *SATAN* are useful for finding application-level vulnerabilities.

The best simulated attacks are those conducted by an outside agency. The external group has no agenda to protect local staff or users; in fact they may well be strongly motivated to find vulnerabilities, to demonstrate their effectiveness.

Recommendation 8
Provide digital signature-capable email tools.

Digital signatures are not often required, but the facility should always be available so that in cases where it is important it can be used. Most systems that provide digital signatures also offer message encryption, which is less commonly needed but can also be useful.

Recommendation 9
Provide multiple independent public key servers.

In cases where asymmetric authentication systems are employed (SSH-based authentication or digital signature systems), provide multiple independently managed key servers. No one person should have access to more than one key server. This makes it very difficult to introduce a false key.

8/ CONCLUSION

The Internet is built on an insecure protocol. Despite this, system administrators face the challenge of providing their users with computing environments that ensure the integrity and privacy of their data. Fortunately it is possible to build secure protocols within the insecure ones that are commonly available. Although this approach is not proof against all types of attacks, it offers an effective way to increase security to a satisfactory level. In many cases, presenting an attacker with a challenge is enough to encourage him or her to look to another target.

Most of the tools described, including SSH, Kerberos, and PGP, are available at no cost on the Internet. This paper has presented the motivation for using these tools, the base technology they employ, and the ways in which they can be incorporated into a typical computing environment. The author hopes that this introduction is sufficient to pique the interest and concern of system administrators and their managers and to encourage them to incorporate some of these tools into their own computing sites.

Acknowledgments

The author would like to thank those who took the time to review and comment on this work, including Lonhyn Jasinskyj, Bill Nitzberg, Alan Powers, Jason Thorpe, Dave Tweten, and Jose Zero. John Howard provided the statistical data, without which the threat model section would have been pure speculation, and was also kind enough to review and comment on the paper.

Bibliography

[1] J. D. Howard, *An Analysis of Security Incidents on the Internet 1989-1995*, PhD Dissertation at Carnegie Mellon University, Pittsburgh, PA, Copyright ©1997.
Also `http://www.cert.org/research/JHThesis/index.html`

[2] "Security of the Internet," from *The Froehlich/Kent Encyclopedia of Telecommunications*, vol. 15, pp. 231-255. Published by Marcel Dekker, New York, Copyright ©1997.
Also `http://www.cert.org/encyc_article/tocencyc.html`

[3] B. Schneier, *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*, Copyright ©1996 by John Wiley & Sons, Inc.

[4] McKusick, Bostic, Karels, and Quarterman, *The Design and Implementation of the 4.4BSD Operating System*, Copyright ©1996 by Addison-Wesley Publishing Company, Inc.

[5] Steiner, Neuman, and Schiller, "Kerberos: An Authentication Service for Open Network Systems," Technical Report, MIT Athena, March 30, 1988.

[6] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," Internet RFC-1510, September 1993. Also `ftp://ftp.math.utah.edu/pub/rfc/rfc1510.txt`.

[7] Bellovin and Merritt, "Limitations of the Kerberos Authentication System," *Proceedings of the Winter 1991 USENIX Conference*, pp. 253-267.

[8] Rinne, Ylonen, Kivinen, Saarinen, and Lehtinen, "SSH Protocol Architecture," Internet-Draft of 6 August 1998 (this is a work-in-progress). Also
`http://ietf.org/internet-drafts/draft-ietf-secsh-architecture-02.txt`

[9] Rinne, Ylonen, Kivinen, Saarinen, and Lehtinen, "SSH Authentication Protocol," Internet-Draft of 6 August 1998 (this is a work-in-progress). Also
`http://ietf.org/internet-drafts/draft-ietf-secsh-userauth-04.txt`

[10] Rinne, Ylonen, Kivinen, Saarinen, and Lehtinen, "SSH Connection Protocol," Internet-Draft of 6 August 1998 (this is a work-in-progress). Also
`http://ietf.org/internet-drafts/draft-ietf-secsh-connect-04.txt`

[11] Rinne, Ylonen, Kivinen, Saarinen, and Lehtinen, "SSH Transport Layer Protocol," Internet-Draft of 6 August 1998 (this is a work-in-progress). Also
`http://ietf.org/internet-drafts/draft-ietf-secsh-transport-04.txt`

[12] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," Internet-Draft of July 1998 (this is a work-in-progress). Also `http://ietf.org/internet-drafts/draft-ietf-ipsec-arch-sec-07.txt`

[13] S. Kent and R. Atkinson, "IP Authentication Header (AH)," Internet-Draft of July 1998 (this is a work-in-progress). Also `http://ietf.org/internet-drafts/draft-ietf-ipsec-auth-header-07.txt`

[14] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," Internet-Draft of July 1998 (this is a work-in-progress). Also `http://ietf.org/internet-drafts/draft-ietf-ipsec-esp-v2-06.txt`

[15] N. Haller, "The S/Key One-time Password System," *Proceedings of the Symposium on Network and Distributed Systems Security*, San Diego CA, Feb. 1994.

[16] D. McDonald, R. Atkinson, C. Metz, "One-Time Passwords in Everything (OPIE): Experiences with Building and Using Stronger Authentication," *Proceedings of the fifth USENIX UNIX Security Symposium*, pp. 177-186, Salt Lake City, Utah, June 1995.

[17] Rivest, Shamir, and Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 26, No. 1, pp. 96-99, Jan. 1983.

[18] B. S. Kaliski, Jr., "The MD2 Message-Digest Algorithm," Internet RFC-1319, April 1992. Also `ftp://ftp.math.utah.edu/pub/rfc/rfc1319.txt`.

[19] N. Rogier and P. Chauvaud, "MD2 is not Secure without the Checksum Byte," *Design, Codes, and Cryptography*, Vol. 12, No. 3, pp. 245-251, Nov. 1997.

[20] H. Dobbertin, "Alf Swindles Ann," *Cryptobytes*, Vol. 1, No. 3, 5, 1995.

[21] National Institute of Standards and Technology (NIST), *FIPS Publication 180: Secure Hash Standard (SHS)*, May 1993.

[22] National Institute of Standards and Technology (NIST), *Announcement of Weakness in the Secure Hash Standard*, May 1994.

[23] R. Rivest, "The MD4 Message-Digest Algorithm," Internet RFC-1320, April 1992. Also `ftp://ftp.math.utah.edu/pub/rfc/rfc1320.txt`.

[24] R. Rivest, "The MD5 Message-Digest Algorithm," Internet RFC-1321, April 1992. Also `ftp://ftp.math.utah.edu/pub/rfc/rfc1321.txt`.

[25] National Bureau of Standards, *Data Encryption Standard, DES*, (FIPS-46), published by the Government Printing Office, Washington D.C., 1977.

[26] R. Rivest, "A Description of the RC2(r) Encryption Algorithm," Internet RFC-2268, January 1998. Also `ftp://ftp.math.utah.edu/pub/rfc/rfc2268.txt`.

[27] R. Rivest, "The IDEA Encryption Algorithm," *Dr. Dobb's Journal of Software Tools*, Vol. 18, No. 13, December 1993, pp. 50,52,54,56,106.

[28] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, Nov. 1976, pp. 74-84.

[29] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, Feb. 1978, pp. 120-126.

[30] R. Rivest, A. Shamir, L. Adleman, "On Digital Signatures and Public Key Cryptosystems," MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, Jan. 1979.

[31] Security Dynamics Technologies, Inc., "A Discussion on the Issues Related to Smart Card Introduction," marketing white-paper. Also `http://www.securitydynamics.com/products/whitepapers/smtcard.html`.

[32] Security Dynamics Technologies, Inc., "SecurID(r) Authentication Tokens," product datasheet. Also `http://www.securitydynamics.com/products/datasheets/tokens.html`.

[33] P. Zimmerman, *PGP(tm) User's Guide*, Vol. I and II, revised 11 October 1994, from PGP v. 2.6.2 distribution.

[34] P. Zimmerman, *The Official PGP User's Guide*, published by MIT Press Copyright ©1995.

[35] J. Bradley and N. Davies, "Analysis of the SSL Protocol," Technical Report, Department of Computer Science, University of Bristol, Number CSTR-95-021, June 1995. Also `http://www.cs.bris.ac.uk/Tools/Reports/Abstracts/1995-bradley.html`.

[36] J. Bradley, "The SSL Reference Implementation Project," Master's Thesis, Department of Computer Science, University of Bristol, October 1995. Also `http://www.cs.bris.ac.uk/Tools/Reports/Abstracts/1995-bradley-0.html`.

[37] W. Rosenberry, D. Kenney, and G. Fisher, *Understanding DCE*, Copyright ©1992 O'Reilly & Associates.

[38] W. Hu, *DCE Security Programming*, Copyright ©1995 O'Reilly and Associates, Inc.